Эссе по теме "Криптографические хеш-функции"

Костиков Егор Вячеславович

Апрель 2024

Содержание

1. Введение

В данной работе приводится описание основных архитектур, используемых при построении современных хеш-функций (Sponge-конструкция, конструкция HAIFA), а также приводятся описания реализации следующих хеш-функций: MD5, SHA-1, SHA-256, SHA-3, ГОСТ Р 34.11-94, ГОСТ 34.11-2018 (Стрибог), BLAKE-256, RIPEMD-256, Whirlpool, Fast Syndrome Based Hash (FSB). Также приводится более подробное описание и анализ хеш-функции RIPEMD-256 (на основе известных результатов анализа RIPEMD-128).

2. Термины и определения

Криптографической функцией хеширования (хеш-функцией) называется отображение $H : V^* \to V_n$, где n — натуральное число, V^* — множество всех двоичных векторов конечной размерности (включая пустую строку), V_n — множество всех n-мерных двоичных векторов. Возвращаемое хеш-функцией значение называется хеш-значением.

Коллизионная атака на хеш-функцию — поиск двух различных входных блоков криптографической хеш-функции, производящих одинаковые значения хеш-функции.

Далее в работе используются следующие обозначения:

- $X <<< n$ — циклический сдвиг содержимого X на n бит влево;
- $X >>> n$ — циклический сдвиг содержимого X на n бит вправо;
- $X << n$ — сдвиг содержимого X на n бит влево (с дописыванием нулевого бита);
- $X >> n$ — сдвиг содержимого X на n бит вправо (с дописыванием нулевого бита);
- $x \parallel y$ — конкатенация строк x и y;
- $x \oplus y$ — побитовое сложение x и y;
- $GF(q)$ — конечное поле Галуа, содержащее q элементов.

3. Описание архитектур хеш-функций

3.1. Sponge-конструкция

Конструкция Sponge была впервые представлена в 2007 году в работе [1].

"Губка"состоит из внутреннего состояния S, содержащее данные фиксированного размера в b бит. Состояние разбивается на две части: первая часть имеет размер r, вторая — c ($b = r + c$). Изначально все биты состояния инициализируются нулями. Входное сообщение M дополняется некоторой функцией pad, после выполнения которой длина дополненного сообщения становится кратна заданной длине блока сообщения, и разбивается на r-битовые блоки. Также в работе "губки"участвует внутренняя b-битная функция f. Работа "губки"состоит из двух фаз:

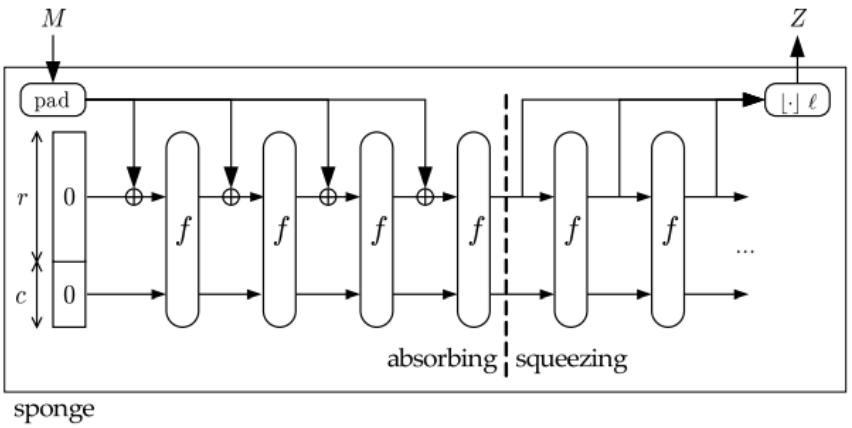
- Фаза "впитывания"(absorbing phase)

  На данном этапе r-битные блоки исходного сообщения M подвергаются операции побитового сложения с первой частью состояния, результат сохраняется в r-битовой части состояния, после чего все обновленное состояние поступает на вход функции f. Когда все блоки сообщения обработаны подобным образом, то "губка"переходит в фазу "выжимания".

- Фаза "выжимания"(squeezing phase)

  На данном этапе первая часть состояния возвращается в качестве выходных блоков, после чего все состояние поступает на вход функции f. Количество операций определяется запрошенным количеством бит l.

После последней итерации вывод обрезается до первых l бит.



Конструкция sponge [2]

Стоит отметить, что хеш-функции, построенные на основе конструкции sponge могут быть отличены от случайного оракула только за счет внутренних коллизий, что свидетельствует о высокой степени сходства с идеальным случайным оракулом и обеспечивает надежность и безопасность применения sponge-функций.

3.2. Конструкция HAIFA

Конструкция HAIFA (The HAsh Iterative FrAmework) была впервые предложена в 2007 году в работе [3]. Конструкция HAIFA заменяет традиционную конструкцию Меркла-Дамгарда. HAIFA улучшает безопасность хеш-функций, поддерживает определение семейств хеш-функций и переменный размер хеш-значения.

Стандартная конструкция Меркла-Дамгарда заключается в следующем:

- Пусть на вход поступило сообщение M, разбитое на k n-битовых блоков $M = M_1 \parallel ... \parallel M_k$; пусть также определено некоторое значение m_c-битного вектора инициализации $h_0 = IV$.
- Пусть определена функция сжатия $C = C(h, M)$, принимающая m_c-битное значение h и n-битное значение M.
- Далее k раз производятся обращения к функции сжатия: $h_i = C(h_{i-1}, M_i)$, $i = 1, ..., k$;
- Полученное на k-ой итерации значение h_k возвращается в качестве хеш-значения сообщения M.

В конструкции HAIFA функция сжатия имеет следующий вид: $C = C(h, M, bits, salt)$, где h — m_c-битный блок, M — n-битный блок, $bits$ — количество бит, захешированных к этому времени, $salt$ — используемая соль.

В HAIFA обязательным шагом при хешировании является расширение сообщения: при этом используется следующая схема расширения:

1) Сообщение дополняется одним единичным битом;
2) Сообщение дополняется необходимым числом нулевых битов, чтобы длина дополняемого сообщения стала сравнима по модулю n с $n - (t + r)$.
3) Сообщение расширяется за счет приписывания t-битового значения длины сообщения;
4) Сообщение расширяется за счет приписывания r-битового значения хешированного блока.

В HAIFA реализована возможность изменять размер хеша. Пусть IV — начальное значение, выбранное разработчиком функции сжатия, пусть m — требуемая длина выхода. Для создания хеш-функции из m бит вычисляется следующее начальное значение: $IV_m = C(IV, m, 0, 0)$. В результате получается значение m, "закодированное"первыми r битами, за которым идет единичный бит и $n - r - 1$ нулевых битов. После того, как посчитался последний блок, итоговым значением являются m бит последнего значения функции сжатия цепочки.

Таким образом, для хеширования сообщения необходимо проделать следующие действия:

- Расширить сообщение так, как было описано ранее;
- Подсчитать значение IV_m так, как было описано ранее;
- В цикле обработать все блоки дополненного сообщения, вычисляя значение функции сжатия: $h_i = C(h_{i-1}, M_i, bits, salt)$;
- В случае необходимости обрезать последнее полученное значение до нужного числа бит.

Конструкция HAIFA обеспечивает улучшенную стойкость по сравнению с традиционной конструкцией Меркла-Дамгарда. Добавление параметров, таких как количество захешированных бит и значение соли, помогает предотвратить различные виды атак, такие как атаки поиска первого и второго прообраза. Кроме того, HAIFA поддерживает различные режимы итерации для функций сжатия, что позволяет адаптировать хеш-функции под конкретные требования безопасности и производительности.

Одним из недостатков HAIFA является возможность использования предварительных вычислений для обхода безопасности, если длина соли ($salt$) недостаточно велика. Рекомендуется, чтобы длина соли составляла как минимум 64 бита или, по возможности, как минимум $\frac{m_c}{2}$ бит, чтобы предотвратить такие атаки. Также отмечается, что для некоторых атак HAIFA требует более длинную соль, чем другие конструкции, что может оказать плохое влияние на производительность.

4. Описания хеш-функций

4.1. MD5

Описание алгоритма работы хеш-функции MD5 приводится в соответствии со стандартом [4]. На вход алгоритма поступает двоичное сообщение $m = m_0 m_1 ... m_{b-1}$ длины $b \geqslant 0$ бит; на выходе алгоритм выдает 128-битный хеш входного сообщения. Опишем шаги алгоритма:

1) Расширение сообщения

   Сообщение расширяется путем добавления в конец сообщения битов по следующему правилу:

   а) В конец сообщения добавляется один единичный бит;

   б) После этого в конец сообщения добавляются нулевые биты до тех пор, пока длина сообщения не станет сравнима с 448 по модулю 512.

2) Добавление длины

   Сообщение дополнительно расширяется путем добавления в конец сообщения 64-битного представления длины сообщения b. При этом, если $b > 2^{64}$, то добавляются только младшие 64 бита двоичного представления b. Данные биты добавляются как два 32-битных слова, причем сначала добавляются младшие 32 бита, а затем старшие.

   После данных преобразований длина сообщения оказывается кратной 512.

3) Инициализация буфера

   При вычислении значения хеш-функции используются четыре 32-битных регистра A, B, C и D, которые перед началом работы инициализируются следующими 16-ричными значениями:

   - A = $0x01234567$;
   - B = $0x89ABCDEF$;
   - C = $0xFEDCBA98$;
   - D = $0x76543210$.

4) Обработка сообщения

   Определим четыре вспомогательных функции $F(X,Y,Z), G(X,Y,Z), H(X,Y,Z), I(X,Y,Z)$, принимающих на вход три 32-битных слова и возвращающих одно 32-битное значение:

   - $F(X,Y,Z) = (X \wedge Y) \vee (\bar{X} \wedge Z)$;
   - $G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \bar{Z})$;
   - $H(X,Y,Z) = X \oplus Y \oplus Z$;
   - $I(X,Y,Z) = Y \oplus (X \vee \bar{Z})$.

   Определим вспомогательную таблицу $T = T[1]...T[64]$, состоящую из 64 элементов, где $T[i] = int|4294967296 \cdot \sin(i)|$, где int — функция взятия целой части.

   Обработка исходного сообщения происходит в цикле блоками по 512 бит — на новой итерации очередной блок помещается во вспомогательный массив $X = X[0]...X[15]$, состоящий из шестнадцати 32-битных слов. Предварительно сохраняются значения регистров A, B, C и D после предыдущей итерации (или начальные значения для первой итерации): AA = A, BB = B, CC = C, DD = D. Выполняются четыре раунда, в каждом из которых совершаются шестнадцать однотипных операций:

   - Первый раунд
     Обозначим через $[abcd\ k\ s\ i]$ следующую операцию: $a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)$.
   - Второй раунд
     Обозначим через $[abcd\ k\ s\ i]$ следующую операцию: $a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s)$.

```
[ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
[ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
[ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]
```

Операции первого раунда

```
[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
[ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
[ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
[ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]
```

Операции второго раунда

```
[ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
[ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
[ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
[ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]
```

Операции третьего раунда

- Третий раунд
  Обозначим через $[abcd\ k\ s\ i]$ следующую операцию: $a = b + ((a + H(b, c, d) + X[k] + T[i])\ <<<\ s)$.
- Четвертый раунд
  Обозначим через $[abcd\ k\ s\ i]$ следующую операцию: $a = b + ((a + I(b, c, d) + X[k] + T[i])\ <<<\ s)$.

```
[ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
[ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
[ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
[ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]
```

Операции четвертого раунда

- Действия по окончании итерации: A = AA + A; B = BB + B; C = CC + C; D = DD + D.

5) Результат вычислений

   Результатом вычисления хеш-функции MD5 является 128-битное значение A||B||C||D, составленное из содержимого соответствующих регистров после последней итерации.

4.2. SHA-1

Описание хеш-функции SHA-1 приводится в соответствии с текстом стандарта [5]. На вход алгоритма поступает произвольное двоичное сообщение, состоящее из не более чем $2^{64} - 1$ бит; на выходе алгоритм выдает 160-битное хеш-значение входного сообщения. Опишем алгоритм SHA-1:

1) Расширение сообщения

   Сообщение расширяется путем добавления в конец сообщения битов по следующему правилу:

   а) В конец сообщения добавляется один единичный бит;

   б) После этого в конец сообщения добавляются нулевые биты до тех пор, пока длина сообщения не станет сравнима с 448 по модулю 512.

2) Добавление длины

   Сообщение дополнительно расширяется путем добавления в конец сообщения 64-битного представления длины сообщения. Данные биты добавляются как два 32-битных слова, причем сначала добавляются старшие 32 бита, а затем младшие.

   После данных преобразований длина сообщения оказывается кратной 512.

3) Инициализация буфера

   При вычислении значения хеш-функции используются пять 32-битных регистра H_0, H_1, H_2, H_3 и H_4, которые перед началом работы инициализируются следующими 16-ричными значениями:

   - $H_0 = 0x67452301$;
   - $H_1 = 0xEFCDAB89$;
   - $H_2 = 0x98BADCFE$;
   - $H_3 = 0x10325476$;
   - $H_4 = 0xC3D2E1F0$.

4) Обработка сообщения

   Определим четыре вспомогательные параметрические функции, принимающие и возвращающие 32-битные значения:

   - $f_t(B,C,D) = (B \wedge C) \vee (\bar{B} \wedge D)$, где $0 \leqslant t \leqslant 19$;
   - $f_t(B,C,D) = B \oplus C \oplus D$, где $20 \leqslant t \leqslant 39$;
   - $f_t(B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$, где $40 \leqslant t \leqslant 59$;
   - $f_t(B,C,D) = B \oplus C \oplus D$, где $60 \leqslant t \leqslant 79$.

   Также определим следующие константы:

   - $K_t = 0x5A827999$, где $0 \leqslant t \leqslant 19$;
   - $K_t = 0x6ED9EBA1$, где $20 \leqslant t \leqslant 39$;
   - $K_t = 0x8F1BBCDC$, где $40 \leqslant t \leqslant 59$;
   - $K_t = 0xCA62C1D6$, где $60 \leqslant t \leqslant 79$.

   Обработка исходного сообщения происходит в цикле блоками по 512 бит — на новой итерации очередной блок помещается во вспомогательный массив $W = W[0]...W[15]$, состоящий из шестнадцати 32-битных слов. Очередная итерация состоит из следующих действий:

   а) $W[t] = (W[t-3] \oplus W[t-8] \oplus W[t-14] \oplus W[t-16]) <<< 1$, для $16 \leqslant t \leqslant 79$;

   б) Сохраняются значения вспомогательных регистров: $A = H_0$; $B = H_1$; $C = H_2$; $D = H_3$; $E = H_4$.

   в) $For\ t = 0\ to\ 79\ do$ {
   $TEMP = A <<< 5 + f_t(B,C,D) + E + W[t] + K_t$;
   $E = D; D = C; C = B <<< 30; B = A; A = TEMP$;
   }

   г) Действия по окончании итерации: $H_0 = H_0 + A$; $H_1 = H_1 + B$; $H_2 = H_2 + C$; $H_3 = H_3 + D$; $H_4 = H_4 + E$.

5) Результат вычислений

   Результатом вычисления хеш-функции SHA-1 является 160-битное значение $H_0||H_1||H_2||H_3||H_4$, составленное из содержимого соответствующих регистров после последней итерации.

4.3. SHA-256

Описание алгоритма SHA-256 приводится в соответствии с описанием, приведенным в стандарте [6]. На вход алгоритма поступает произвольное двоичное сообщение, состоящее из не более чем $2^{64} - 1$ бит; на выходе алгоритм выдает 256-битный хеш входного сообщения. Опишем шаги данного алгоритма:

1) Расширение сообщения

   Сообщение расширяется путем добавления в конец сообщения битов по следующему правилу:

   а) В конец сообщения добавляется один единичный бит;

б) После этого в конец сообщения добавляются нулевые биты до тех пор, пока длина сообщения не станет сравнима с 448 по модулю 512.

2) Добавление длины

Сообщение дополнительно расширяется путем добавления в конец сообщения 64-битного представления длины сообщения. Данные биты добавляются как два 32-битных слова, причем сначала добавляются старшие 32 бита, а затем младшие.

После данных преобразований длина сообщения оказывается кратной 512.

3) Инициализация буфера

При вычислении значения хеш-функции используются восемь 32-битных регистра $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$, которые перед началом работы инициализируются следующими 16-ричными значениями:

- $H_0 = 0x6A09E667$;
- $H_1 = 0xBB67AE85$;
- $H_2 = 0x3C6EF372$;
- $H_3 = 0xA54FF53A$;
- $H_4 = 0x510E527F$;
- $H_5 = 0x9B05688C$;
- $H_6 = 0x1F83D9AB$;
- $H_7 = 0x5BE0CD19$.

4) Обработка сообщения

Определим шесть вспомогательных функций, принимающих и возвращающих 32-битные значения:

- $Ch(x, y, z) = (x \wedge y) \oplus (\bar{x} \wedge z)$;
- $Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$;
- $\sigma_0(x) = (x >>> 7) \oplus (x >>> 18) \oplus (x >> 3)$, где через $>>$ обозначена операция нециклического сдвига содержимого x;
- $\sigma_1(x) = (x >>> 17) \oplus (x >>> 19) \oplus (x >> 10)$;
- $S_0(x) = (x >>> 2) \oplus (x >>> 13) \oplus (x >>> 22)$;
- $S_1(x) = (x >>> 6) \oplus (x >>> 11) \oplus (x >> 25)$.

Определим вспомогательные шестнадцатеричные константы $K_0, ..., K_{63}$ в соответствии с Таблицей 1 (увеличение индекса слева направо, сверху вниз).

428a2f98	71374491	b5c0fbcf	e9b5dba5	3956c25b	59f111f1	923f82a4	ab1c5ed5
d807aa98	12835b01	243185be	550c7dc3	72be5d74	80deb1fe	9bdc06a7	c19bf174
e49b69c1	efbe4786	0fc19dc6	240ca1cc	2de92c6f	4a7484aa	5cb0a9dc	76f988da
983e5152	a831c66d	b00327c8	bf597fc7	c6e00bf3	d5a79147	06ca6351	14292967
27b70a85	2e1b2138	4d2c6dfc	53380d13	650a7354	766a0abb	81c2c92e	92722c85
a2bfe8a1	a81a664b	c24b8b70	c76c51a3	d192e819	d6990624	f40e3585	106aa070
19a4c116	1e376c08	2748774c	34b0bcb5	391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
748f82ee	78a5636f	84c87814	8cc70208	90befffa	a4506ceb	bef9a3f7	c67178f2

Таблица 1: Таблица значений констант $K_0, ..., K_{63}$

Обработка исходного сообщения происходит в цикле блоками по 512 бит — на новой итерации очередной блок помещается во вспомогательный массив $W = W[0]...W[15]$, состоящий из шестнадцати 32-битных слов. Очередная итерация состоит из следующих действий:

а) $W[t] = \sigma_1(W[t-2]) + W[t-7] + \sigma_0(W[t-15]) + W[t-16]$, для $16 \leqslant t \leqslant 63$;

б) Сохраняются значения вспомогательных регистров: $A = H_0$; $B = H_1$; $C = H_2$; $D = H_3$; $E = H_4$; $F = H_5$; $G = H_6$; $H = H_7$.

в) $For\ t = 0\ to\ 63\ do$ {
$TEMP_1 = H + S_1(E) + Ch(E,F,G) + K_t + W[t]$;
$TEMP_2 = S_0(A) + Maj(A,B,C)$;
$H = G; G = F; F = E; E = D + TEMP_1; D = C; C = B; B = A; A = TEMP_1 + TEMP_2$;
}

г) Действия по окончании итерации: $H_0 = H_0 + A$; $H_1 = H_1 + B$; $H_2 = H_2 + C$; $H_3 = H_3 + D$; $H_4 = H_4 + E$; $H_5 = H_5 + F$; $H_6 = H_6 + G$; $H_7 = H_7 + H$.

5) Результат вычислений

Результатом вычисления хеш-функции SHA-256 является 256-битное значение $H_0 \| H_1 \| H_2 \| H_3 \| H_4 \| H_5 \| H_6 \| H_7$, составленное из содержимого соответствующих регистров после последней итерации.

4.4. SHA-3

Описание хеш-функции SHA-3 приводится в соответствии со стандартом [7]. В стандарте SHA-3 для произвольного входного сообщения M допускается подсчет хеш-значения, имеющего длину 224, 256, 384 или 512 бит. Приведем описание данного алгоритма:

Алгоритм SHA-3 построен на основе sponge-конструкции; в алгоритме используется состояние, которое может быть представлено в виде трехмерного массива $A = A[x,y,z]$ из $5x5xw$ бит, где $w = 64$ бита (возможны другие значения $w = 2^l$ для некоторого натурального l).

Вычисление хеш-значения основано на использовании функции $KECCAK$, построение которой приведено далее; при этом длина хеш-значения может варьироваться:

- SHA3-224(M) = $KECCAK[448](M \| 01, 224)$;
- SHA3-256(M) = $KECCAK[512](M \| 01, 256)$;
- SHA3-384(M) = $KECCAK[768](M \| 01, 384)$;
- SHA3-512(M) = $KECCAK[1024](M \| 01, 512)$;

Опишем используемые в алгоритме преобразования (для каждого преобразования входной массив будем обозначать через A, а выходной — через A'):

1) Преобразование $\theta = \theta(A)$:

- Для всех $0 \leqslant x < 5, 0 \leqslant z < w$:
$C[x,z] = A[x,0,z] \oplus A[x,1,z] \oplus A[x,2,z] \oplus A[x,3,z] \oplus A[x,4,z]$;
- Для всех $0 \leqslant x < 5, 0 \leqslant z < w$:
$D[x,z] = C[(x-1) \mod 5, z] \oplus C[(x+1) \mod 5, (z-1) \mod w]$;
- Для всех $0 \leqslant x < 5, 0 \leqslant y < 5, 0 \leqslant z < w$:
$A'[x,y,z] = A[x,y,z] \oplus D[x,z]$.

2) Преобразование $\rho = \rho(A)$:

- Для всех $0 \leqslant z < w$:
$A'[0,0,z] = A[0,0,z]$;
- Пусть $x = 1, y = 0$;
- $For\ t = 0\ to\ 23\ do$ {
Для всех $0 \leqslant z < w$: $A'[x,y,z] = A[x,y,(z - \frac{(t+1)(t+2)}{2}) \mod w]$;
$x = y, y = (2x + 3y) \mod 5$;
}

3) Преобразование $\pi = \pi(A)$:

   - Для всех $0 \leqslant x < 5, 0 \leqslant y < 5, 0 \leqslant z < w$:
     $A'[x, y, z] = A[(x + 3y) \mod 5, x, z]$;

4) Преобразование $\chi = \chi(A)$:

   - Для всех $0 \leqslant x < 5, 0 \leqslant y < 5, 0 \leqslant z < w$:
     $A'[x, y, z] = A[x, y, z] \oplus (A[(x + 1) \mod 5, y, z] \cdot A[(x + 2) \mod 5, y, z])$;

5) Преобразование $rc = rc(t)$. Пусть t — целое число.

   а) Если $t \mod 255 = 0$, то $rc(t) = 1$;

   б) Пусть $R = 10000000$;

   в) $For\ i = 1\ to\ t \mod 255\ do$ {
   $R = 0 \parallel R$;
   $R[0] = R[0] \oplus R[8]$;
   $R[4] = R[4] \oplus R[8]$;
   $R[5] = R[5] \oplus R[8]$;
   $R[6] = R[6] \oplus R[8]$;
   $R = R[0] \parallel ... \parallel R[7]$;
   }

   г) $rc(t) = R[0]$.

6) Преобразование $\iota = \iota(A, i_r)$, где i_r — номер раунда.

   а) Для всех $0 \leqslant x < 5, 0 \leqslant y < 5, 0 \leqslant z < w$: $A'[x, y, z] = A[x, y, z]$;

   б) $RC = 0^w$;

   в) $RC[2^j - 1] = rc(j + 7i_r), j = 0, ..., l$;

   г) Для всех $0 \leqslant z < w$: $A'[0, 0, z] = A'[0, 0, z] \oplus RC[z]$;

   д) $\iota(A, i_r) = A'$.

7) Определим раундовую функцию $Rnd(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$, где i_r — номер раунда.

8) Определим функцию $KECCAK\text{-}p[b, n_r](S)$, где b — длина строки S, n_r — число раундов.

   а) Строка S переводится в массив A по правилу: $A[x, y, z] = S[w(5y + x) + z]$;

   б) $A = Rnd(A, i_r), i_r = 12 + 2l - n_r, ..., 12 + 2l - 1$;

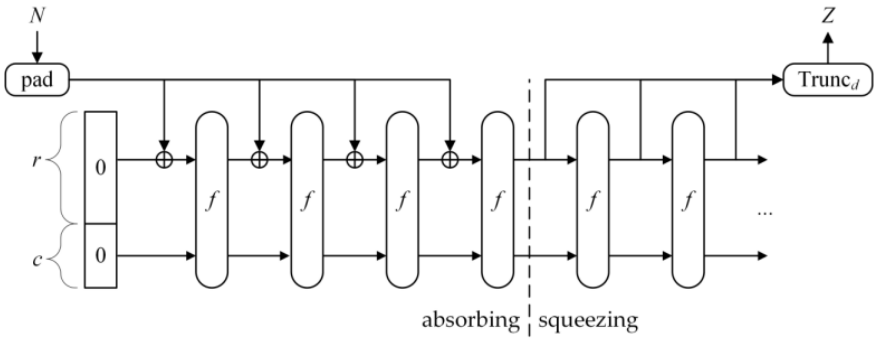
   в) Массив A переводится в строку S' длины b;

   г) $KECCAK\text{-}p[b, n_r](S) = S'$.

   В алгоритме строится классическая sponge-конструкция, изображенная на схеме далее и обозначаемая как $SPONGE[f, pad, r](N, d)$.

   На схеме в качестве внутренней функции f используется построенная функция $KECCAK\text{-}p[b, n_r](S)$.

   Итоговое преобразование, используемое в работе SHA-3 может быть записано в виде:
   $KECCAK[c](N, d) = SPONGE[KECCAK\text{-}p[1600, 24], pad10*1, 1600 - c](N, d)$, где $pad10*1$ — некоторая функция расширения входного сообщения.



sponge-конструкция SHA-3

4.5. ГОСТ Р 34.11-94

Описание алгоритма ГОСТ Р 34.11-94 приводится в соответствии с описанием, приведенным в стандарте [8]. На вход алгоритма поступает произвольное двоичное сообщение; на выходе алгоритм выдает 256-битный хеш входного сообщения. Приведем описание работы данного алгоритма:

1) При подсчете хеш-значения используется специальная шаговая функция хеширования $\chi = \chi(M, H)$ (отображение, которое преобразует два 256-битных блока данных в один), алгоритм вычисления которой включает в себя три части:

   а) Генерация ключей — 256-битных слов K_1, K_2, K_3 и K_4.

   Пусть $X = x_4 \| x_3 \| x_2 \| x_1 = y_{32} \| \ldots \| y_1$ — 256-битное слово, x_1, x_2, x_3 и x_4 — 64-битные слова, образующие X, а $y_1, \ldots, y_{32}$ — 8-битные слова, образующие X. Тогда определим преобразования $A(X) = (x_1 \oplus x_2) \| x_4 \| x_3 \| x_2$ и $P(X) = y_{\varphi(32)} \| \ldots \| y_{\varphi(1)}$, где $\varphi(i + 1 + 4 \cdot (k - 1)) = 8 \cdot i + k$, где $i = 0, \ldots, 3$; $k = 1, \ldots, 8$.

   Пусть на вход шаговой функции поступили 256-битные слова H и M. Тогда для вычисления ключей будут выполнены следующие действия:

   - $TEMP_1 = H, TEMP_2 = M, K_1 = P(TEMP_1 \oplus TEMP_2)$;
   - Для $i = 2, 3, 4$: $TEMP_1 = A(TEMP_1) \oplus C_i, TEMP_2 = A(A(TEMP_2)), K_i = P(TEMP_1 \oplus TEMP_2)$, где $C_2 = 0, C_3 = 0xff00ffff000000ffff0000ff00ffff0000ff00ff00ff00ffff00ff00ff00ff00, C_4 = 0$.

   б) Шифрующее преобразование

   Пусть на вход шаговой функции поступили 256-битные слова H и M. На данном этапе осуществляется зашифрование 64-битных подслов слова $H = h_4 \| h_3 \| h_2 \| h_1$ на сгенерированных ключах K_1, K_2, K_3 и K_4.

   Шифрование происходит следующим образом: $s_i = E_{K_i}(h_i)$, где $i = 1, 2, 3, 4$, через E_{K_i} обозначен алгоритм шифрования 64-битного блока на ключе K_i по ГОСТ 28147 в режиме простой замены. В результате образуется 256-битное слово $S = s_4 \| s_3 \| s_2 s_1$.

   в) Перемешивающее преобразование

   Определим функцию ψ, принимающую 256-битный блок $X = x_{16} \| \ldots \| x_1$ и возвращающую значение $\psi(X) = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_{13} \oplus x_{16} \| x_{16} \| \ldots \| x_2$.

   Таким образом, в качестве значения шаговой функции хеширования χ, принимающей 256-битные блоки M и H, определяется значение $\chi(M, H) = \psi^{61}(H \oplus \psi(M \oplus \psi^{12}(S)))$.

2) Процедура вычисления хеш-функции сообщения $M = m_n \| \ldots \| m_1$, где $m_n, \ldots, m_1$ — разбиение M на 256-битные блоки (длина m_n может быть менее 256 бит), с использованием шаговой функции хеширования состоит из следующих шагов:

   а) Этап 1 — инициализация

   - $H = IV$, где IV — произвольный 256-битный блок, H — текущее значение хеш-функции;
   - $\Sigma = 0^{256}$, где Σ — текущее значение контрольной суммы;
   - $L = 0^{256}$, где L — длина сообщения.

б) Этап 2 — циклическая обработка блоков $m_1, ..., m_{n-1}$; при обработке очередного блока m_i, $i = 1, ..., n-1$, выполняются следующие действия:

- $H = \chi(m_i, H)$;
- $L = L + 256$, где сложение 256-битных блоков происходит по модулю 2^{256};
- $\Sigma = \Sigma + m_i$, где сложение 256-битных блоков происходит по модулю 2^{256}.

в) Этап 3 — обработка финального блока m_n

- $L = L + |m_n|$;
- В начало блока m_n дописываются незначащие нули, пока длина блока не станет равна 256;
- $\Sigma = \Sigma + m_n$, где сложение 256-битных блоков происходит по модулю 2^{256};
- $H = \chi(m_n, H)$;
- $H = \chi(L, H)$;
- $H = \chi(\Sigma, H)$.

Хеш-значением для входного сообщения M является полученное на последнем шаге значение H.

4.6. ГОСТ 34.11-2018 (Стрибог)

Описание алгоритма ГОСТ 34.11 - 2018 (Стрибог) приводится в соответствии с описанием, приведенным в стандарте [9]. На вход алгоритма поступает произвольное двоичное сообщение; на выходе алгоритм выдает 256-битный хеш входного сообщения или 512-битный хеш входного сообщения. Приведем описание работы данного алгоритма:

В работе алгоритма используются следующие константы, таблицы и функции:

1) Векторы инициализации IV: для хеш-функции с длиной хеша в 512 бит $IV = 0^{512}$, для хеш-функции с длиной хеша в 256 бит $IV = (00000001)^{64}$;

2) 512-битные шестнадцатеричные константы $C_1, ..., C_{12}$:

- $C_1 = b1085bda1ecadae9ebcb2f81c0657c1f2f6a76432e45d016714eb88d7585c4fc4b7091926769 01a2422a08a460d31505767436cc744d23dd806559f2a64507$;
- $C_2 = 6fa3b58aa99d2f1a4fe39d460f70b5d7f3feea720a232b9861d55e0f16b501319ab5176b12d699585cb561c2db0aa7ca55dda21bd7cbcd56e679047021b19bb7$;
- $C_3 = f574dcac2bce2fc70a39fc286a3d843506f15e5f529c1f8bf2ea7514b1297b7bd3e20fe490359eb1c1c93a376062db09c2b6f443867adb31991e96f50aba0ab2$;
- $C_4 = ef1fdfb3e81566d2f948e1a05d71e4dd488e857e335c3c7d9d721cad685e353fa9d72c82ed03d675d8b71333935203be3453eaa193e837f1220cbebc84e3d12e$;
- $C_5 = 4bea6bacad4747999a3f410c6ca923637f151c1f1686104a359e35d7800fffbdbfcd1747253af5a3dfff00b723271a167a56a27ea9ea63f5601758fd7c6cfe57$;
- $C_6 = ae4faeae1d3ad3d96fa4c33b7a3039c02d66c4f95142a46c187f9ab49af08ec6cffaa6b71c9ab7b40af21f66c2bec6b6bf71c57236904f35fa68407a46647d6e$;
- $C_7 = f4c70e16eeaac5ec51ac86febf240954399ec6c7e6bf87c9d3473e33197a93c90992abc52d822c3706476983284a05043517454ca23c4af38886564d3a14d493$;
- $C_8 = 9b1f5b424d93c9a703e7aa020c6e41414eb7f8719c36de1e89b4443b4ddbc49af4892bcb929b069069d18d2bd1a5c42f36acc2355951a8d9a47f0dd4bf02e71e$;
- $C_9 = 378f5a541631229b944c9ad8ec165fde3a7d3a1b258942243cd955b7e00d0984800a440bdbb2ceb17b2b8a9aa6079c540e38dc92cb1f2a607261445183235adb$;
- $C_{10} = abbedea680056f52382ae548b2e4f3f38941e71cff8a78db1fffe18a1b3361039fe76702af69334b7a1e6c303b7652f43698fad1153bb6c374b4c7fb98459ced$;
- $C_{11} = 7bcd9ed0efc889fb3002c6cd635afe94d8fa6bbbebab07612001802114846679 8a1d71efea48b9caefbacd1d7d476e98dea2594ac06fd85d6bcaa4cd81f32d1b$;
- $C_{12} = 378ee767f11631bad21380b00449b17acda43c32bcdf1d77f82012d430219f9b5d80ef9d1891cc86e71da4aa88e12852faf417d5d9b21b9948bc924af11bd720$.

3) Перестановка π на множестве; $\{0, ..., 255\}$

252	238	221	17	207	110	49	22	251	196	250	218	35	197	4	77
233	119	240	219	147	46	15	186	23	54	241	187	20	205	95	193
249	24	101	90	226	92	239	33	129	28	60	66	139	1	142	79
5	132	2	174	227	106	143	160	6	11	237	152	127	212	211	31
235	52	44	81	234	200	72	171	242	42	104	162	253	58	206	204
181	112	14	86	8	12	118	18	191	114	19	71	156	183	93	135
21	161	150	41	16	123	154	199	243	145	120	111	157	158	178	177
50	117	25	61	255	53	138	126	109	84	198	128	195	189	13	87
223	245	36	169	62	168	67	201	215	121	214	246	124	34	185	3
224	15	236	222	122	148	176	188	220	232	40	80	78	51	10	74
167	151	96	115	30	0	98	68	26	184	56	130	100	159	38	65
173	69	70	146	39	94	85	47	140	163	165	125	105	213	149	59
7	88	179	64	134	172	29	247	48	55	107	228	136	217	231	137
225	27	131	73	76	63	248	254	141	83	170	144	202	216	133	97
32	113	103	164	45	43	9	91	203	155	37	208	190	229	108	82
89	166	116	210	230	244	180	192	209	102	175	194	57	75	99	182

Таблица 2: Таблица перестановки π

4) Перестановка τ на множестве $\{0, ..., 63\}$;

0	8	16	24	32	40	48	56	1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58	3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60	5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62	7	15	23	31	39	47	55	63

Таблица 3: Таблица перестановки τ

5) Преобразование l, принимающее на вход 64-битный блок данных и заключающееся в умножении справа данного блока на заданную матрицу A над полем $GF(2)$. Строки матрицы A определены в виде шестнадцатеричных констант следующим образом (увеличение номера строки слева направо, сверху вниз):

8e20faa72ba0b470	47107ddd9b505a38	ad08b0e0c3282d1c	d8045870ef14980e
6c022c38f90a4c07	3601161cf205268d	1b8e0b0e798c13c8	83478b07b2468764
a011d380818e8f40	5086e740ce47c920	2843fd2067adea10	14aff010bdd87508
0ad97808d06cb404	05e23c0468365a02	8c711e02341b2d01	46b60f011a83988e
90dab52a387ae76f	486dd4151c3dfdb9	24b86a840e90f0d2	125c354207487869
092e94218d243cba	8a174a9ec8121e5d	4585254f64090fa0	accc9ca9328a8950
9d4df05d5f661451	c0a878a0a1330aa6	60543c50de970553	302a1e286fc58ca7
18150f14b9ec46dd	0c84890ad27623e0	0642ca05693b9f70	0321658cba93c138
86275df09ce8aaa8	439da0784e745554	afc0503c273aa42a	d960281e9d1d5215
e230140fc0802984	71180a8960409a42	b60c05ca30204d21	5b068c651810a89e
456c34887a3805b9	ac361a443d1c8cd2	561b0d22900e4669	2b838811480723ba
9bcf4486248d9f5d	c3e9224312c8c1a0	effa11af0964ee50	f97d86d98a327728
e4fa2054a80b329c	727d102a548b194e	39b008152acb8227	9258048415eb419d
492c024284fbaec0	aa16012142f35760	550b8e9e21f7a530	a48b474f9ef5dc18
70a6a56e2440598e	3853dc371220a247	1ca76e95091051ad	0edd37c48a08a6d8
07e095624504536c	8d70c431ac02a736	c83862965601dd1b	641c314b2b8ee083

Строки матрицы A

6) Функция $S = S(a)$, принимающая 512-битный блок $a = a_{63} \| ... \| a_0$, где a_{63}, ..., a_0 — 8-битные блоки, и выдающая значение $S(a) = \pi(a_{63}) \| ... \| \pi(a_0)$;

7) Функция $P = P(a)$, принимающая 512-битный блок $a = a_{63} \| ... \| a_0$, где a_{63}, ..., a_0 — 8-битные блоки, и выдающая значение $P(a) = a_{\tau(63)} \| ... \| a_{\tau(0)}$;

8) Функция $L = L(a)$, принимающая 512-битный блок $a = a_7 \parallel ... \parallel a_0$, где a_7, ..., a_0 — 64-битные блоки, и выдающая значение $L(a) = l(a_7) \parallel ... \parallel l(a_0)$;

9) Значение хеш-функции сообщения вычисляется с использованием функции сжатия $g_N = g_N(h, m)$, где h, m, N — 512-битные блоки. Функция сжатия возвращает 512-битное значение $g_N(h, m)$, которое определяется следующим образом: $g_N(h, m) = E(LPS(h \oplus N), m) \oplus h \oplus m$, где функция $E = E(K, m) = K_{13} \oplus LPS(K_{12} \oplus ...LPS(K_2 \oplus LPS(m \oplus K_1))...)$, где через $LPS(x)$ обозначен результат последовательного применения функций L, P, S: $LPS(x) = L(P(S(x)))$, K — 512-битный блок.

   Значения K_1, ..., K_{13} определяются следующим образом: $K_1 = K$, $K_i = LPS(K_{i-1} \oplus C_{i-1})$, $i = 2, ..., 13$.

Процедура вычисления хеш-значения для произвольного сообщения $M = m_n \parallel ... \parallel m_1$, где m_n, ..., m_1 — разбиение M на 512-битные блоки (длина m_n может быть мене 512 бит), состоит из следующих этапов:

1) Этап 1 — инициализация

   - $h = IV$;
   - $N = 0^{512}$;
   - $\Sigma = 0^{512}$.

2) Этап 2 — циклическая обработка блоков m_1, ..., m_{n-1}; при обработке очередного блока m_i, $i = 1, ..., n-1$, выполняются следующие действия:

   - $h = g_N(h, m_i)$;
   - $N = N + 512$, где сложение 512-битных блоков происходит по модулю 2^{512};
   - $\Sigma = \Sigma + m_i$, где сложение 512-битных блоков происходит по модулю 2^{512}.

3) Этап 3 — обработка финального блока m_n

   - В начало блока m_n дописывается один единичный бит, после чего дописываются нулевые биты, пока длина блока не станет равна 512 (либо не дописываются, если длина уже равна 512);
   - $h = g_N(h, m_n)$;
   - $N = N + |m_n|$, где $|m_n|$ — длина блока m_n до начала его обработки, сложение происходит по модулю 2^{512};
   - $\Sigma = \Sigma + m_n$, где сложение 512-битных блоков происходит по модулю 2^{512};
   - $h = g_0(h, N)$;
   - $H = g_0(h, \Sigma)$;

   Если для сообщения M необходимо выдать 512-битный хеш, то после всех проделанных операций будет возвращено вычисленное на последнем шаге алгоритма значение H; если необходимо выдать 256-битный хеш, то после всех проделанных операций будут возвращены старшие 256 бит вычисленного на последнем шаге алгоритма значения H.

4.7. BLAKE-256

Описание алгоритма BLAKE-256 приводится на основе спецификации хеш-функции, представленной в [10]. На вход алгоритма поступает произвольное двоичное сообщение длины не более $2^{64} - 1$; на выходе алгоритм выдает 256-битный хеш входного сообщения. Приведем описание работы данного алгоритма:

1) Перед началом работы алгоритма определяются восемь шестнадцатеричных векторов инициализации, шестнадцать шестнадцатеричных констант и десять перестановок на множестве $\{0, ..., 15\}$:

$IV_0 = 6A09E667$	$IV_1 = BB67AE85$	$IV_2 = 3C6EF372$	$IV3 = A54FF53A$
$IV4 = 510E527F$	$IV5 = 9B05688C$	$IV6 = 1F83D9AB$	$IV7 = 5BE0CD19$

Таблица 4: Таблица значений векторов инициализации $IV_0, ..., IV_7$

$c_0 = 243F6A88$	$c_1 = 85A308D3$	$c_2 = 13198A2E$	$c_3 = 03707344$
$c_4 = A4093822$	$c_5 = 299F31D0$	$c_6 = 082EFA98$	$c_7 = EC4E6C89$
$c_8 = 452821E6$	$c_9 = 38D01377$	$c_{10} = BE5466CF$	$c_{11} = 34E90C6C$
$c_{12} = C0AC29B7$	$c_{13} = C97C50DD$	$c_{14} = 3F84D5B5$	$c_{15} = B5470917$

Таблица 5: Таблица значений констант $c_0, ..., c_{15}$

σ_0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
σ_1	14 10 4 8 9 15 13 6 1 12 0 2 11 7 5 3
σ_2	11 8 12 0 5 2 15 13 10 14 3 6 7 1 9 4
σ_3	7 9 3 1 13 12 11 14 2 6 5 10 4 0 15 8
σ_4	9 0 5 7 2 4 10 15 14 1 11 12 6 8 3 13
σ_5	2 12 6 10 0 11 8 3 4 13 7 5 15 14 1 9
σ_6	12 5 1 15 14 13 4 10 0 7 6 3 9 2 8 11
σ_7	13 11 7 14 12 1 3 9 5 0 15 4 8 6 2 10
σ_8	6 15 14 9 11 3 0 8 12 2 13 7 1 4 10 5
σ_9	10 2 8 4 7 6 1 5 15 11 9 14 3 12 13 0

Таблица 6: Перестановки $\sigma_0, ..., \sigma_9$

2) Работа алгоритма основана на функции сжатия $compress = compress(h, m, s, t)$, которая принимает 256-битное значение $h = h_0 \parallel h_7$, 512-битное значение $m = m_0 \parallel ... \parallel m_{15}$, 128-битное значение $s = s_0 \parallel ... \parallel s_3$ и 64-битное значение $t = t_0 \parallel t_1$, где все промежуточные блоки состоят из 32 бит. Данная функция возвращает 256-битное значение $h' = h'_0 \parallel ... \parallel h'_7$.

3) При вычислении значения функции $compress$ используется набор из 16 слов $v_0, ..., v_{15}$, совокупность которых называется состоянием. Инициализация состояния происходит следующим образом:

$v_0 = h_0$	$v_1 = h_1$	$v_2 = h_2$	$v_3 = h_3$
$v_4 = h_4$	$v_5 = h_5$	$v_6 = h_6$	$v_7 = h_7$
$v_8 = s_0 \oplus c_0$	$v_9 = s_1 \oplus c_1$	$v_{10} = s_2 \oplus c_2$	$v_{11} = s_3 \oplus c_3$
$v_{12} = t_0 \oplus c_4$	$v_{13} = t_0 \oplus c_5$	$v_{14} = t_1 \oplus c_6$	$v_{15} = t_1 \oplus c_7$

Таблица 7: Инициализация $v_0, ..., v_{15}$

4) Операции над состоянием $v_0, ..., v_{15}$ производятся посредством серии обращений к раундовой функции $G_i(a, b, c, d)$. После инициализации состояния запускается серия из четырнадцати раундов, каждый из которых состоит из следующих обращений:

$G_0(v_0, v_4, v_8, v_{12})$	$G_1(v_1, v_5, v_9, v_{13})$	$G_2(v_2, v_6, v_{10}, v_{14})$	$G_3(v_3, v_7, v_{11}, v_{15})$
$G_4(v_0, v_5, v_{10}, v_{15})$	$G_5(v_1, v_6, v_{11}, v_{12})$	$G_6(v_2, v_7, v_8, v_{13})$	$G_7(v_3, v_4, v_9, v_{14})$

Таблица 8: Операции в раунде

Во время раунда $0 \leqslant r \leqslant 13$ функцией $G_i(a, b, c, d)$ выполняются следующие действия:

а) $a = a + b + (m_{\sigma_{r \mod 10}(2i)} \oplus c_{\sigma_{r \mod 10}(2i+1)})$;

б) $d = (d \oplus a) >>> 16$;

в) $c = c + d$;

г) $b = (b \oplus c) >>> 12$;

д) $a = a + b + (m_{\sigma_r \bmod 10(2i+1)} \oplus c_{\sigma_r \bmod 10(2i)})$;

е) $d = (d \oplus a) >>> 8$;

ж) $c = c + d$;

з) $b = (b \oplus c) >>> 7$.

После выполнения всех раундов происходят следующие операции, в результате которых определяется выход функции *compress*:

- $h_0^{'} = h_0 \oplus s_0 \oplus v_0 \oplus v_8$;
- $h_1^{'} = h_1 \oplus s_1 \oplus v_1 \oplus v_9$;
- $h_2^{'} = h_2 \oplus s_2 \oplus v_2 \oplus v_{10}$;
- $h_3^{'} = h_3 \oplus s_3 \oplus v_3 \oplus v_{11}$;
- $h_4^{'} = h_4 \oplus s_0 \oplus v_4 \oplus v_{12}$;
- $h_5^{'} = h_5 \oplus s_1 \oplus v_5 \oplus v_{13}$;
- $h_6^{'} = h_6 \oplus s_2 \oplus v_6 \oplus v_{14}$;
- $h_7^{'} = h_7 \oplus s_3 \oplus v_7 \oplus v_{15}$.

5) Хеширование входного сообщения m выполняется следующим образом:

а) Сообщение расширяется путем добавления битов по следующему правилу:

- В конец сообщения приписывается один единичный бит;
- В конец сообщения приписываются нулевые биты до тех пор, пока длина сообщения не станет сравнима с 447 по модулю 512;
- В конец сообщения приписывается еще один единичный бит;
- В конец сообщения приписывается 64-битное представление длины сообщения L.

После данных преобразований длина сообщения оказывается кратной 512.

б) Входное сообщение m делится на 512-битные блоки $m = m_0 \parallel ... \parallel m_{N-1}$. Для каждого блока вычисляется его длина L_i, $0 \leqslant i \leqslant N - 1$, исключая при подсчете те биты, которые были добавлены на предыдущем шаге.

в) Пользователем определяется значение 128-битной соли $s = s_0 \parallel ... \parallel s_3$.

г) Выполняется присваивание $h^0 = IV_0 \parallel ... \parallel IV_7$.

д) В цикле выполняется вызов функции сжатия: $h^{i+1} = compress(h^i, m^i, s, L^i)$, $0 \leqslant i \leqslant N - 1$.

В качестве хеш-значения входного сообщения выдается посчитанное на последней итерации значение h^N.

4.8. RIPEMD-256

4.8.1. Описание хеш-функции

Описание алгоритма RIPEMD-256 приводится на основе работ [11] и [12] разработчиков данной хеш-функции. На вход алгоритма поступает произвольное двоичное сообщение; на выходе алгоритм выдает 256-битный хеш входного сообщения. Опишем работу данного алгоритма:

В работе алгоритма используются следующие функции, константы и таблицы:

1) Четыре параметрические функции:

- $f(j, x, y, z) = x \oplus y \oplus z$, где $0 \leqslant j \leqslant 15$;
- $f(j, x, y, z) = (x \wedge y) \vee (\bar{x} \wedge z)$, где $16 \leqslant j \leqslant 31$;

- $f(j,x,y,z) = (x \vee \bar{y}) \oplus z$, где $32 \leqslant j \leqslant 47$;
- $f(j,x,y,z) = (x \wedge z) \vee (y \wedge \bar{z})$, где $48 \leqslant j \leqslant 63$.

2) Шестнадцатеричные константы:

- $K(j) = 0x00000000$, где $0 \leqslant j \leqslant 15$;
- $K(j) = 0x5A827999$, где $16 \leqslant j \leqslant 31$;
- $K(j) = 0x6ED9EBA1$, где $32 \leqslant j \leqslant 47$;
- $K(j) = 0x8F1BBCDC$, где $48 \leqslant j \leqslant 63$;
- $K^{'}(j) = 0x50A28BE6$, где $0 \leqslant j \leqslant 15$;
- $K^{'}(j) = 0x5C4DD124$, где $16 \leqslant j \leqslant 31$;
- $K^{'}(j) = 0x6D703EF3$, где $32 \leqslant j \leqslant 47$;
- $K^{'}(j) = 0x00000000$, где $48 \leqslant j \leqslant 63$.

3) Начальные значения:

- $h_0 = 0x67452301$;
- $h_1 = 0xEFCDAB89$;
- $h_2 = 0x98BADCFE$;
- $h_3 = 0x10325476$;
- $h_4 = 0x76543210$;
- $h_5 = 0xFEDCBA98$;
- $h_6 = 0x89ABCDEF$;
- $h_7 = 0x01234567$.

4) Таблицы $s = s_i$ и $s^{'} = s_i^{'}$, $0 \leqslant i \leqslant 63$, определяющие величину выполняемого циклического сдвига (изменение индекса слева направо, сверху вниз):

11,	14,	15,	12,	5,	8,	7,	9,	11,	13,	14,	15,	6,	7,	9,	8
7,	6,	8,	13,	11,	9,	7,	15,	7,	12,	15,	9,	11,	7,	13,	12
11,	13,	6,	7,	14,	9,	13,	15,	14,	8,	13,	6,	5,	12,	7,	5
11,	12,	14,	15,	14,	15,	9,	8,	9,	14,	5,	6,	8,	6,	5,	12

Таблица 9: Таблица s

8,	9,	9,	11,	13,	15,	15,	5,	7,	7,	8,	11,	14,	14,	12,	6
9,	13,	15,	7,	12,	8,	9,	11,	7,	7,	12,	7,	6,	15,	13,	11
9,	7,	15,	11,	8,	6,	6,	14,	12,	13,	5,	14,	13,	13,	7,	5
15,	5,	8,	11,	14,	14,	6,	14,	6,	9,	12,	9,	12,	5,	15,	8

Таблица 10: Таблица $s^{'}$

5) Таблицы $r = r_i$ и $r^{'} = r_i^{'}$, $0 \leqslant i \leqslant 63$, определяющие ... (изменение индекса слева направо, сверху вниз):

0,	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11,	12,	13,	14,	15
7,	4,	13,	1,	10,	6,	15,	3,	12,	0,	9,	5,	2,	14,	11,	8
3,	10,	14,	4,	9,	15,	8,	1,	2,	7,	0,	6,	13,	11,	5,	12
1,	9,	11,	10,	0,	8,	12,	4,	13,	3,	7,	15,	14,	5,	6,	2

Таблица 11: Таблица r

5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12
6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2
15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13
8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14

Таблица 12: Таблица $r^{'}$

Перед началом вычисления хеш-значения сообщение расширяется путем добавления битов по следующему правилу:

1) В конец сообщения приписывается один единичный бит;
2) В конец сообщения приписываются нулевые биты до тех пор, пока длина сообщения не станет сравнима с 448 по модулю 512;
3) В конец сообщения приписывается 64-битное представление длины сообщения (если исходная длина сообщения превышает 2^{64}, то от ее битовой длины используется только младшие 64 бит).

После данных преобразований длина сообщения оказывается кратной 512.

После описанных преобразований расширенное входное сообщение X разделяется на t 256-битных блоков: $X = X[0] \| ... \| X[t-1]$. Каждый блок $X[i]$ при этом подразбивается на 16 32-битных блоков: $X[i] = X[i][0] \| ... \| X[i][15]$, $0 \leqslant i \leqslant i - 1$. Алгоритм вычисления хеш-значения (сложения производятся по модулю 2^{32}):

$For\ i = 0\ to\ t - 1\ do$ {
  $A = h_0; B = h_1; C = h_2; D = h_3;$
  $A^{'} = h_4; B^{'} = h_5; C^{'} = h_6; D^{'} = h_7;$
  $For\ j = 0\ to\ 63\ do$ {
    $TEMP = (A + f(j, B, C, D) + X[i][r_j] + K(j)) <<< s_j;$
    $A = D; D = C; C = B; B = TEMP;$
    $TEMP = (A + f(63 - j, B^{'}, C^{'}, D^{'}) + X[i][r_j^{'}] + K^{'}(j)) <<< s_j^{'};$
    $A^{'} = D^{'}; D^{'} = C^{'}; C^{'} = B^{'}; B^{'} = TEMP;$
    Если $j = 15$, то значения A и $A^{'}$ меняются местами;
    Если $j = 31$, то значения B и $B^{'}$ меняются местами;
    Если $j = 47$, то значения C и $C^{'}$ меняются местами;
    Если $j = 63$, то значения D и $D^{'}$ меняются местами;
  }
  $h_0 = h_0 + A; h_1 = h_1 + B; h_2 = h_2 + C; h_3 = h_3 + D;$
  $h_4 = h_4 + A^{'}; h_5 = h_5 + B^{'}; h_6 = h_6 + C^{'}; h_7 = h_7 + D^{'};$
}

Итоговым хеш-значением для входного сообщения является полученное после последней итерации 256-битное значение $h_0 \| ... \| h_7$.

4.8.2. Анализ хеш-функции

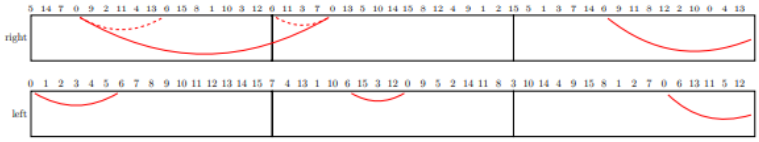
Хеш-функция RIPEMD-256 является расширением хеш-функции RIPEMD-128, которая выдает хеш-значение длиной в 128 бит. RIPEMD-256 предназначена для применения в приложениях, которым требуется более длинное значение хеш-функции без необходимости более высокого уровня безопасности, чем обеспечиваемый RIPEMD-128 (хотя удвоение хеш-значения и уменьшает вероятность коллизий). Хеш-функция RIPEMD-256 сконструирована на основе RIPEMD-128 путем инициализации двух параллельных потоков вычисления с разными начальными значениями и исключения комбинации двух потоков после последнего шага преобразования обновления состояния. Кроме того, после каждого раунда между двумя параллельными потоками вычислений происходит обмен некоторыми внутренними переменными состояния, чтобы сделать два потока зависимыми друг от друга.

Так как хеш-функция RIPEMD-256 не является более криптостойкой по сравнению с RIPEMD-128, и так как работ по анализу RIPEMD-256 в литературе практически не представлено, то далее будут описаны теоретические атаки, применимые к RIPEMD-128. Стоит также отметить, что эксплуатируемых на практике уязвимостей данного алгоритма на текущий момент не обнаружено.

В работе [13] была представлена атака на 38-раундовый RIPEMD-128, в результате которой со сложностью 2^{14} для хеш-функции была найдена коллизия. Предлагаемая авторами статьи стратегия поиска коллизии состоит из трех шагов:

1) Поиск стартовой точки — нахождение такой стартовой настройки (различия только в нескольких конкретных словах сообщений), которая с высокой вероятностью может привести к дифференциальной характеристике после 15 шага.

   Поскольку RIPEMD-128 имеет два потока с разной перестановкой слов сообщения, первым шагом в атаке является определение тех слов сообщения, которые могут содержать различия, учитывая несколько ограничений, позволяющих эффективно провести всю атаку в целом. Прежде всего, авторы стремятся к получению дифференциальных характеристик с высокой вероятностью после 15 шага в обоих потоках (линиях) вычислений. Такие дифференциальные характеристики с высокой вероятностью могут быть построены, если различия, вносимые словами сообщения, немедленно устраняются с помощью локальных коллизий, охватывающих всего несколько шагов. Авторы делают акцент на рассмотрении двух слов сообщения: m_0 и m_6 ($M = m_0 \parallel \ldots \parallel m_{15}$), за счет чего объединением в одной линии двух локальных коллизий (что приводит к одной длительной локальной коллизии между этапами 3 и 20) удается к третьему раунду коллизию для 38 шагов RIPEMD-128.



Локальные коллизии при использовании слов m_0 и m_6

2) Поиск дифференциальной характеристики

   После того, как была определена стартовая точка (то есть слова сообщения, которые могут содержать различия), авторы предлагают использовать автоматизированный инструмент для поиска дифференциальной характеристик с высокой вероятностью. Автоматическое устройство поиска основано на подходе, описанном в работе [14], для поиска сложных нелинейных дифференциальных характеристик и подтверждающих пар сообщений для SHA-2. Основная идея заключается в рассмотрении дифференциальных характеристик, накладывающих произвольные условия на пары битов с использованием так называемых обобщенных условий. Обобщенные условия основаны на различиях в знаках битов и учитывают все 16 возможных условий для пар битов. Автоматическое устройство позволяет находить сложные нелинейные дифференциальные характеристики и решать нелинейные уравнения, включающие условия на слова состояния и свободные биты сообщения, то есть находить подтверждающие пары сообщений. Стоит отметить, что разница в сообщении не фиксируется до начала поиска, чтобы инструмент мог найти оптимальное решение. Сначала поиск происходит в первой линии — с учетом наличия 14 ограничений на возможные значения участвующих в цепочке значений сложность поиска составляет примерно 2^{14}. Аналогичный поиск происходит и во второй линии.

3) Поиск пары сообщений

   Чтобы выполнить все условия, налагаемые дифференциальной характеристикой в первом раунде, необходимо применить методы модификации сообщений. Авторы также с использованием автоматизированного инструмента добиваются модификации сообщения на первых 16 шагах. Также сложность модификации сообщения может быть компенсирована путем рандомизации, например, слова сообщения m_{12}, чтобы найти решение также для характеристики с высокой вероятностью во втором раунде (и в третьем раунде). Авторы отмечают, что на практике, используя приблизительно 2^{30} возможных значений для m_{12}, можно найти решение для дифференциальной характеристики (сложность 2^{14} после первого раунда), включая изменение сообщения, менее чем за секунду.

Таким образом, со сложностью 2^{14} была построена коллизия для 38-шаговой хеш-функции RIPEMD-128.

M_1	9431bddf 7b9827d6 f54a64a9 df41a58a fd707a50 dad10eb6 48b0cc76 be66cb8c ab3b7afa 084ba98e ab0a4798 2a4b0d06 a79bf8b7 3fd6008a 4da2112d 849c5b9c
M_2	952bc70f d0840848 eafffa57 0ca3c38a 45383ffb ddc6a9a1 796f1e20 0b9ff55f ddb80113 f0ffe1b5 b7d75dc0 82c7298f f2c442f4 96cbf293 c441d662 06e9eec2
M_2^*	952bc50e d0840848 eafffa57 0ca3c38a 45383ffb ddc6a9a1 79ef1e21 0b9ff55f ddb80113 f0ffe1b5 b7d75dc0 82c7298f f2c442f4 96cbf293 c441d662 06e9eec2
ΔM_2	00000201 00000000 00000000 00000000 00000000 00000000 00800001 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
H_2	a0a00507 fd4c7274 ba230d53 87a0d10a
H_2^*	a0a00507 fd4c7274 ba230d53 87a0d10a
ΔH_2	00000000 00000000 00000000 00000000

Построенная коллизия M_2 и M_2^*

В статье [15] представлена атака на 40-раундовый RIPEMD-128, в результате которой со сложность 2^{35} для хеш-функции была найдена коллизия. Коллизия достигается путем построения 512-битных блоков $N \parallel M$ и $N \parallel M'$, где N, M, M' — 256-битные блоки. В статье предложен следующий алгоритм поиска коллизии состоит из следующих шагов:

Вычисления в RIPEMD-128 (как и в RIPEMD-256) происходят параллельно в рамках двух линий, условно обозначаемых как $line_1$ и $line_2$ (в описании алгоритма RIPEMD-256 это результаты первого в цикле циклического сдвига и второго соответственно).

1) Авторы выбирают различия в сообщениях следующим образом: пусть $M = m_0 \parallel ... \parallel m_{15}$; $\Delta M = M' - M$ выбирается так, чтобы $\Delta m_i = 0, 0 \leqslant i \leqslant 15, i \neq 2, i \neq 12$; $\Delta m_2 = 2^8$, $\Delta m_{12} = -2$. На основе данных условий определяются дифференциальные характеристики для $line_1$ и $line_2$. Выбор обоснован тем, что он приводит к возникновению локальной коллизии с 25 шага по 29 шага в операциях $line_1$ и локальной коллизии между 6 и 32 шагами в операциях $line_2$. Это свойство делает возможность проведения коллизионной атаки на 40-шаговый RIPEMD-128, используя различия в сообщениях, удовлетворяющих указанным свойствам.

2) Авторами выводятся наборы достаточных условий, которые обеспечивают выполнение дифференциальных характеристик. Для данных условий устанавливается, что вероятность их выполнения приблизительно равна 2^{-35}.

3) Алгоритм поиска коллизий состоит из следующих шагов:

   а) Необходимо случайно выбрать блоки m_i, $0 \leqslant i \leqslant 15$, $i \neq 12$, после чего их необходимо модифицировать так, чтобы был выполнен набор достаточных условий $line_1$, а также чтобы набор достаточных условий $line_2$ выполнялся с вероятностью 2^{-35};

   б) Если достаточные условия $line_2$ оказались не выполнены, то необходимо перейти к шагу а);

   в) Необходимо случайно выбрать блок m_{12} и посчитать хеш-значения получившихся M и M' 40-шаговым RIPEMD-128. Если значения совпали, то выдать M и M'. Иначе необходимо перейти к шагу а).

Таким образом, с временной сложностью примерно $2^{35} + 2^{41}$ удается осуществить коллизионную атаку на 40-шаговую вариацию хеш-функции RIPEMD-128. Пример построения коллизии (для данных сообщений $\Delta m_2 = 2^8, \Delta m_{12} = -2$):

N	664504b6	d6e949ba	2176407d	85426fc1	5ec28995	c3d318b	787db431	ae2c13fb
	cee9d90	c5078e4b	84bae5bc	99f3f4ae	d7403dc6	917fa14c	85155db5	fd9311e6
M	a7e4a89f	6278156c	2a535118	90eba965	670841b2	ea6f8dcb	800766d9	d0bfa5c6
	ffe74d8e	6df2c5f7	a3ffdbfd	53e156d4	54f75d	f0d3a13f	7eef12b9	ef317f76
M'	a7e4a89f	6278156c	2a535218	90eba965	670841b2	ea6f8dcb	800766d9	d0bfa5c6
	ffe74d8e	6df2c5f7	a3ffdbfd	53e156d4	54f75b	f0d3a13f	7eef12b9	ef317f76
H	a76df6ab	43ae1a6e	171d9fda	da03925e				

Построенная коллизия $N \parallel M$ и $N \parallel M'$

4.9. Whirlpool

Описание хеш-функции Whirlpool приводится на основе статьи [16] разработчиков данной хеш-функции. На вход хеш-функции подается произвольное двоичное сообщение длины не более $2^{256} - 1$ бит; на выходе функция выдает 512-битный хеш входного сообщения. Опишем работу данного алгоритма:

Хеш-функция основана на специальном блочном шифре, который работает с 512-битными данными. Промежуточный результат вычисления хеш-значения называется состоянием, которое представляется квадратной матрицей состояния из восьми строк и восьми столбцов из элементов поля $GF(2^8)$. Пространство таких матриц будем далее обозначать через $M_{8x8}[GF(2^8)]$.

- Преобразование μ, преобразующее 512-битные блоки в обозначенные матрицы. Пусть $a \in GF(2^8)^{64}, b \in M_{8x8}[GF(2^8)]$. Тогда $\mu(a) = b \iff b_{i,j} = a_{8i+j}, 0 \leqslant i, j \leqslant 7$.
- Преобразование γ. Пусть $a, b \in M_{8x8}[GF(2^8)]$. Тогда $\gamma(a) = b \iff b_{i,j} = S[a_{i,j}], 0 \leqslant i, j \leqslant 7$.

	00_x	01_x	02_x	03_x	04_x	05_x	06_x	07_x	08_x	09_x	$0A_x$	$0B_x$	$0c_x$	$0d_x$	$0E_x$	$0F_x$
00_x	18_x	23_x	$c6_x$	$E8_x$	87_x	$B8_x$	01_x	$4F_x$	36_x	$A6_x$	$d2_x$	$F5_x$	79_x	$6F_x$	91_x	52_x
10_x	60_x	Bc_x	$9B_x$	$8E_x$	$A3_x$	$0c_x$	$7B_x$	35_x	$1d_x$	$E0_x$	$d7_x$	$c2_x$	$2E_x$	$4B_x$	FE_x	57_x
20_x	15_x	77_x	37_x	$E5_x$	$9F_x$	$F0_x$	$4A_x$	dA_x	58_x	$c9_x$	29_x	$0A_x$	$B1_x$	$A0_x$	$6B_x$	85_x
30_x	Bd_x	$5d_x$	10_x	$F4_x$	cB_x	$3E_x$	05_x	67_x	$E4_x$	27_x	41_x	$8B_x$	$A7_x$	$7d_x$	95_x	$d8_x$
40_x	FB_x	EE_x	$7c_x$	66_x	dd_x	17_x	47_x	$9E_x$	cA_x	$2d_x$	BF_x	07_x	Ad_x	$5A_x$	83_x	33_x
50_x	63_x	02_x	AA_x	71_x	$c8_x$	19_x	49_x	$d9_x$	$F2_x$	$E3_x$	$5B_x$	88_x	$9A_x$	26_x	32_x	$B0_x$
60_x	$E9_x$	$0F_x$	$d5_x$	80_x	BE_x	cd_x	34_x	48_x	FF_x	$7A_x$	90_x	$5F_x$	20_x	68_x	$1A_x$	AE_x
70_x	$B4_x$	54_x	93_x	22_x	64_x	$F1_x$	73_x	12_x	40_x	08_x	$c3_x$	Ec_x	dB_x	$A1_x$	$8d_x$	$3d_x$
80_x	97_x	00_x	cF_x	$2B_x$	76_x	82_x	$d6_x$	$1B_x$	$B5_x$	AF_x	$6A_x$	50_x	45_x	$F3_x$	30_x	EF_x
90_x	$3F_x$	55_x	$A2_x$	EA_x	65_x	BA_x	$2F_x$	$c0_x$	dE_x	$1c_x$	Fd_x	$4d_x$	92_x	75_x	06_x	$8A_x$
$A0_x$	$B2_x$	$E6_x$	$0E_x$	$1F_x$	62_x	$d4_x$	$A8_x$	96_x	$F9_x$	$c5_x$	25_x	59_x	84_x	72_x	39_x	$4c_x$
$B0_x$	$5E_x$	78_x	38_x	$8c_x$	$d1_x$	$A5_x$	$E2_x$	61_x	$B3_x$	21_x	$9c_x$	$1E_x$	43_x	$c7_x$	Fc_x	04_x
$c0_x$	51_x	99_x	$6d_x$	$0d_x$	FA_x	dF_x	$7E_x$	24_x	$3B_x$	AB_x	cE_x	11_x	$8F_x$	$4E_x$	$B7_x$	EB_x
$d0_x$	$3c_x$	81_x	94_x	$F7_x$	$B9_x$	13_x	$2c_x$	$d3_x$	$E7_x$	$6E_x$	$c4_x$	03_x	56_x	44_x	$7F_x$	$A9_x$
$E0_x$	$2A_x$	BB_x	$c1_x$	53_x	dc_x	$0B_x$	$9d_x$	$6c_x$	31_x	74_x	$F6_x$	46_x	Ac_x	89_x	14_x	$E1_x$
$F0_x$	16_x	$3A_x$	69_x	09_x	70_x	$B6_x$	$d0_x$	Ed_x	cc_x	42_x	98_x	$A4_x$	28_x	$5c_x$	$F8_x$	86_x

Таблица преобразования S

- Преобразование π. Пусть $a, b \in M_{8x8}[GF(2^8)]$. Тогда $\pi(a) = b \iff b_{i,j} = a_{(i-j) \mod 8, j}, 0 \leqslant i, j \leqslant 7$.
- Преобразование θ. Пусть $a, b \in M_{8x8}[GF(2^8)]$. Тогда $\theta(a) = b \iff b = a \cdot C$.

$$\begin{bmatrix} 01_x & 01_x & 04_x & 01_x & 08_x & 05_x & 02_x & 09_x \\ 09_x & 01_x & 01_x & 04_x & 01_x & 08_x & 05_x & 02_x \\ 02_x & 09_x & 01_x & 01_x & 04_x & 01_x & 08_x & 05_x \\ 05_x & 02_x & 09_x & 01_x & 01_x & 04_x & 01_x & 08_x \\ 08_x & 05_x & 02_x & 09_x & 01_x & 01_x & 04_x & 01_x \\ 01_x & 08_x & 05_x & 02_x & 09_x & 01_x & 01_x & 04_x \\ 04_x & 01_x & 08_x & 05_x & 02_x & 09_x & 01_x & 01_x \\ 01_x & 04_x & 01_x & 08_x & 05_x & 02_x & 09_x & 01_x \end{bmatrix}$$

Матрица C

- Преобразование $\sigma[k]$. Пусть $a, b, k \in M_{8x8}[GF(2^8)]$. Тогда $\sigma[k](a) = b \iff b_{i,j} = a_{i,j} \oplus k_{i,j}, 0 \leqslant i, j \leqslant 7$.
- Раундовые константы c^r. Пусть $c^r \in M_{8x8}[GF(2^8)], r > 0$ — номер раунда. Тогда:

- $c^r_{0,j} = S[8(r-1)+j], 0 \leqslant j \leqslant 7$;
- $c^r_{i,j} = 0, 1 \leqslant i \leqslant 7, 0 \leqslant j \leqslant 7$.

- Раундовая функция $\rho[k]$. Пусть $a, b, k \in M_{8x8}[GF(2^8)]$. Тогда для каждого раунда r раундовая функция имеет вид: $\rho[k](a) \equiv \sigma[k](\theta(\pi(\gamma(a))))$.
- Процедура расширения ключа. 512-битный ключ $K \in M_{8x8}[GF(2^8)]$ расширяется до последовательности раундовых ключей $K^0, ..., K^R$ следующим образом:
  - $K^0 = K$;
  - $K^r = \rho[c^r](K^{r-1}), r > 0$.

  Как правило, число раундов R по умолчанию равно 10.
- Внутренний блочный шифр $W[K]$. Пусть $a, b, K \in M_{8x8}[GF(2^8)]$. Тогда шифром выполняется следующая последовательность преобразований: $W[K](a) = \rho[K^r](\rho[K^{r-1}](...\rho[K^1](\sigma[K^0](a))...))$.

Процедура подсчета хеш-значения:

1) К концу сообщения M дописывается единичный бит;
2) К концу сообщения M дописываются нулевые биты, чтобы длина полученного сообщения была кратная 256 нечетное число раз;
3) К концу сообщения M дописывается 256-битное значение длины сообщения.
4) Обработка сообщения:
   а) Дополненное сообщение разбивается на 512-битные блоки $m_t \parallel ... \parallel m_1$.
   б) $\eta_i = \mu(m_i)$;
   в) $H_0 = \mu(IV)$, где $IV = 0^{512}$;
   г) $H_i = W[H_{i-1}](\eta_i) \oplus H_{i-1} \oplus \eta_i, 1 \leqslant i \leqslant t$.

В качестве хеш-значение сообщения M возвращается значение $\mu^{-1}(H_t)$.

4.10. Fast Syndrome Based Hash (FSB)

Описание хеш-функции Fast Syndrome Based Hash приводится в соответствии с работой [17] разработчиков данной хеш-функции.

Алгоритм FSB основан на использовании структуры Меркла—Дамгарда, использует для вычисления функции сжатия случайную двоичную матрицу H и имеет несколько параметров:

- n — число столбцов матрицы H;
- r — число строк матрицы H и число бит на выходе хеш-функции;
- w — число столбцов матрицы H, выбираемых из матрицы при подсчете значения функции сжатия;
- s — число бит на входе функции сжатия;

Для данных параметров должны выполняться условия: $s > r, s = w \log_2(\frac{n}{w})$.

Под словом веса w и длины n будем понимать n-битный блок данных, в котором содержится ровно w единичных бит. Слово веса w и длины n является регулярным, если в каждом интервале от $(i-1)\frac{n}{w}$ до $i\frac{n}{w}$ содержится ровно один ненулевой элемент, $1 \leqslant i \leqslant w$.

Алгоритм вычисления функции сжатия следующий:

1) Матрица H разделятся на w подматриц из r строк и $\frac{n}{w}$ столбцов;
2) Входное сообщение из s бит разделяется на части $s_1, ..., s_w$ длины $\log_2(\frac{n}{w})$;
3) Каждое значение $s_1, ..., s_w$ преобразуется в целое десятичное число от 1 до $\frac{n}{w}$;
4) Для каждого значения i из подматрицы H_i выбирается соответствующий десятичному значению s_i столбец;
5) Выбранные w столбцов побитово складываются, в результате чего образуется выход функции сжатия длины r.

5. Заключение

В работе были описаны основные архитектуры построения хеш-функций: Sponge-конструкция и конструкция HAIFA, а также были рассмотрены хеш-функции MD5, SHA-1, SHA-256, SHA-3, ГОСТ Р 34.11-94, ГОСТ 34.11-2018 (Стрибог), BLAKE-256, RIPEMD-256, Whirlpool и Fast Syndrome Based Hash (FSB), построенные в том числе на основе указанных ранее конструкций. Также более подробно была рассмотрена хеш-функция RIPEMD-256. Данная хеш-функция является модернизацией хеш-функции RIPEMD-128 и предназначена для применения в приложениях, которым требуется более длинное значение хеш-значения без необходимости более высокого уровня безопасности, чем обеспечиваемый RIPEMD-128. Уязвимостей для алгоритма RIPEMD-256 на текущий момент не обнаружено. Вместе с тем известны теоретические атаки на ограниченные по раундам версии алгоритма RIPEMD-128, позволяющие совершить на алгоритм коллизионную атаку, что теоретически (в силу определенной преемственности алгоритмов) может быть перенесено на алгоритм RIPEMD-256. Также стоит отметить, что RIPEMD-256, очевидно, за счет увеличения длины хеш-значения теоретически является более стойким к коллизионной атаке, чем RIPEMD-128. Таким образом, лучшей атакой на RIPEMD-256 на текущей момент является атака путем полного перебора.

Список литературы

1. Sponge Functions / G. Bertoni [и др.] // ECRYPT Hash Workshop 2007. — 2007.
2. Cryptographic sponge functions / G. Bertoni [и др.]. — 2011.
3. *Biham E., Dunkelman O.* A Framework for Iterative Hash Functions — HAIFA // Second NIST Cryptographic Hash Workshop. — 2007.
4. *Rivest R.* The MD5 Message-Digest Algorithm // RFC 1321. — 1992.
5. SECURE HASH STANDARD (SHS) (FIPS PUB 180-1) // Federal Information Processing Standards Publication 180-1 (FIPS PUB 180-1). — 1995.
6. SECURE HASH STANDARD (SHS) (FIPS PUB 180-2) // Federal Information Processing Standards Publication 180-2 (FIPS PUB 180-2). — 2002.
7. FIPS PUB 202 - SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. — 2015.
8. ГОСТ Р 34.11-94 «Информационная технология. Криптографическая защита информации. Функция хэширования». — 1994.
9. ГОСТ 34.11-2018 «Информационная технология. Криптографическая защита информации. Функция хэширования». — 2018.
10. *Aumasson J., Henzen L., Meier W.* Sha-3 proposal blake. — 2008.
11. *Dobbertin H., Bosselaers A., Preneel B.* The hash function RIPEMD-160. Dedicated webpage. — 1996. — URL: http://homes.esat.kuleuven.be/~bosselae/ripemd160.html.
12. — RIPEMD-160, a strengthened version of RIPEMD // FSE 1996. Lecture Notes in Computer Science, vol 1039. — 1996.
13. *Mendel F., Nad T., Schläffer M.* Collision Attacks on the Reduced Dual-Stream Hash Function RIPEMD-128 // Fast Software Encryption. FSE 2012. — 2012.
14. *Mendel F., Nad T., Schl ¨affer M.* Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions // ASIACRYPT 2011. — 2011.
15. *Wang G.* Practical collision attack on 40-step RIPEMD-128 // Topics in Cryptology – CT-RSA 2014. — 2014.
16. *Barreto P., Rijmen V.* The WHIRLPOOL Hashing Function. — 2003.
17. *Augot D., Finiasz M., Sendrier N.* A Fast Provably Secure Cryptographic Hash Function. — 2003.